In [1]:
```
from pathlib import Path
from typing import List, Tuple, Dict
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import (
    confusion_matrix,
    ConfusionMatrixDisplay,
    roc_curve,
    auc,
    precision_recall_curve,
    PrecisionRecallDisplay,
    RocCurveDisplay,
)
from minisom import MiniSom
```

In [2]:
```
# trains a SOM on the input data
def fit_som(data: np.ndarray, grid: Tuple[int, int] = (22, 22), seed: int = 42) ->
    rows, cols = grid  # SOM grid dimensions set to 22x22
    som = MiniSom(
        x=rows, y=cols,
        input_len=data.shape[1],         # number of features
        sigma=3.0,                       # spread of the neighborhood
        learning_rate=0.5,               # speed of learning
        neighborhood_function="gaussian",# type of neighborhood function
        random_seed=seed                 # for reproducibility
    )
    som.random_weights_init(data)
    som.train_batch(data, num_iteration=10_000, verbose=False)  # train the SOM
    return som

# create a lookup table from each SOM node BMU to a majority label
def majority_vote_lookup(som: MiniSom, data: np.ndarray, labels: np.ndarray) -> Dic
    vote: Dict[Tuple[int, int], List[int]] = {}

    for vec, lbl in zip(data, labels):                  # go through each data poin
        bmu = som.winner(vec)                           # find best-matching unit (
        vote.setdefault(bmu, []).append(lbl)            # collect all labels that m

    # assign each BMU the most common label (rounded average)
    return {bmu: int(round(np.mean(v))) for bmu, v in vote.items()}

# predict labels for new data using the trained SOM and the vote map
def predict_som(som: MiniSom, vote_map: Dict[Tuple[int, int], int], data: np.ndarra
    # for each input vector, find its BMU and use the vote_map to assign a predicte
    return np.array([vote_map.get(som.winner(v), 0) for v in data])
```

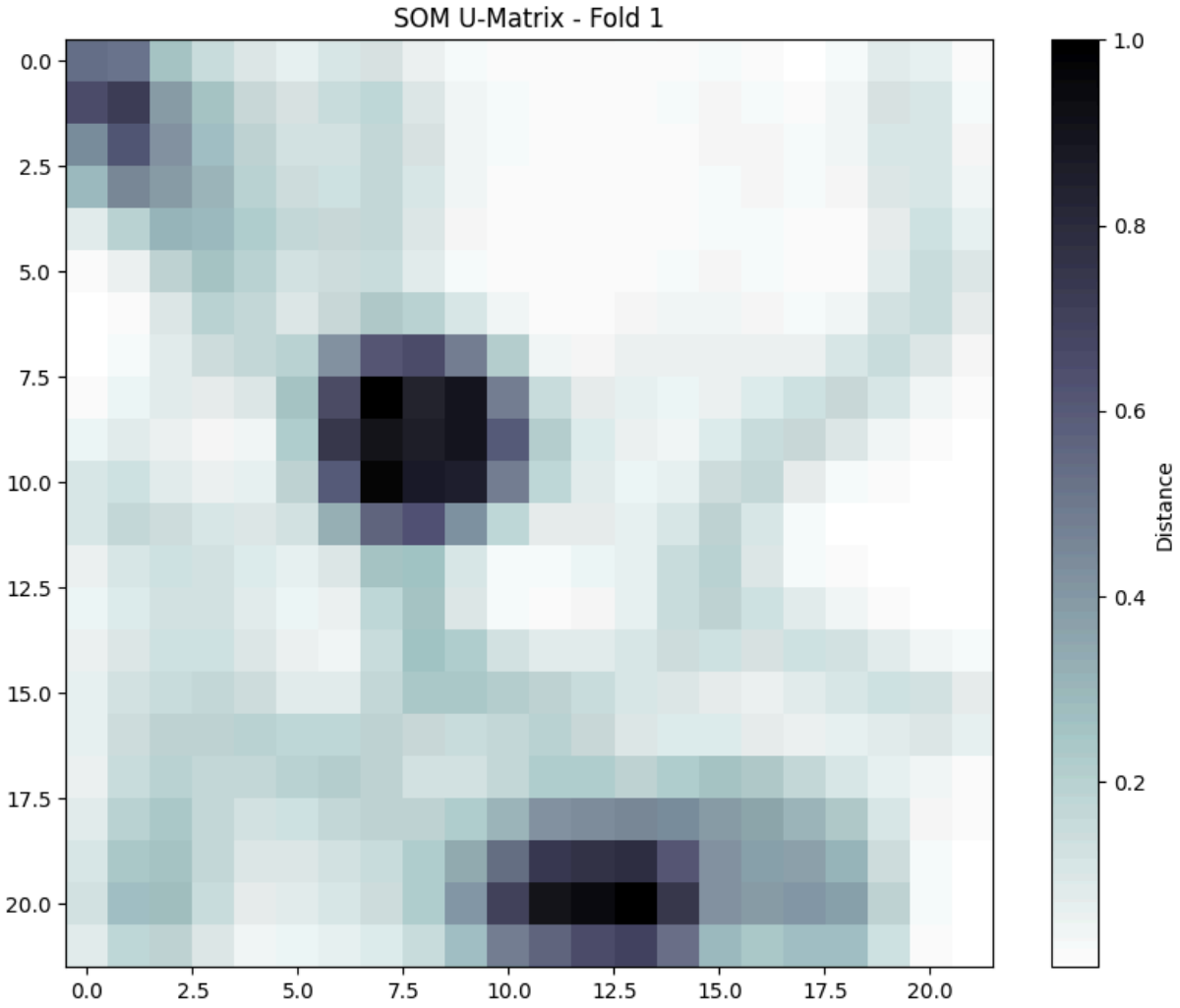
In [3]:
```
import time
import psutil
import os


# performs cross-validation using a Self-Organizing Map (SOM)
```

```python
def som_cross_validate(Syn_df: pd.DataFrame, feature_columns: List[str], grid: Tupl
    accuracies = []
    process = psutil.Process(os.getpid())
    oof_true, oof_score = [], []

    # resource monitoring starting
    overall_start_time = time.time()
    overall_start_ram = process.memory_info().rss / 1024 / 1024  # in MB
    overall_start_cpu = psutil.cpu_percent(interval=1)

    # loop over each fold in the dataset
    for fold in sorted(Syn_df["Fold"].unique()):
        train_df = Syn_df[Syn_df["Fold"] != fold]
        validate_df = Syn_df[Syn_df["Fold"] == fold]

        scaler = StandardScaler()
        X_train = scaler.fit_transform(train_df[feature_columns])
        X_validate = scaler.transform(validate_df[feature_columns])
        y_train = train_df["Label"].values
        y_validate = validate_df["Label"].values

        som = fit_som(X_train, grid)
        vote_map = majority_vote_lookup(som, X_train, y_train)
        y_predict = predict_som(som, vote_map, X_validate)

        # plot U-Matrix for each fold
        plt.figure(figsize=(10, 8))
        u_matrix = som.distance_map()
        plt.imshow(u_matrix, cmap='bone_r')
        plt.colorbar(label='Distance')
        plt.title(f'SOM U-Matrix - Fold {fold+1}')
        plt.savefig(f"som_umatrix_fold_{fold+1}.png", dpi=300, bbox_inches="tight")
        plt.show()
        plt.close()

        # accuracy
        acc = (y_predict == y_validate).mean()
        accuracies.append(float(acc))
        print(f"Fold {fold+1}: accuracy = {acc:.4f}")

        dists  = np.array([som.quantization_error(np.array([v])) for v in X_validat
        scores = -dists                          # higher = “more like map (attack)”

        oof_true.extend(y_validate.tolist())
        oof_score.extend(scores.tolist())

    y_bin = (np.array(oof_score) > 0).astype(int)

    cm = confusion_matrix(oof_true, y_bin)
    ConfusionMatrixDisplay(confusion_matrix=cm).plot(cmap="Blues")
    plt.title("SOM Confusion Matrix")
    plt.show()

    fpr, tpr, _ = roc_curve(oof_true, oof_score)
    RocCurveDisplay(fpr=fpr, tpr=tpr,
                    roc_auc=auc(fpr, tpr)).plot()
```

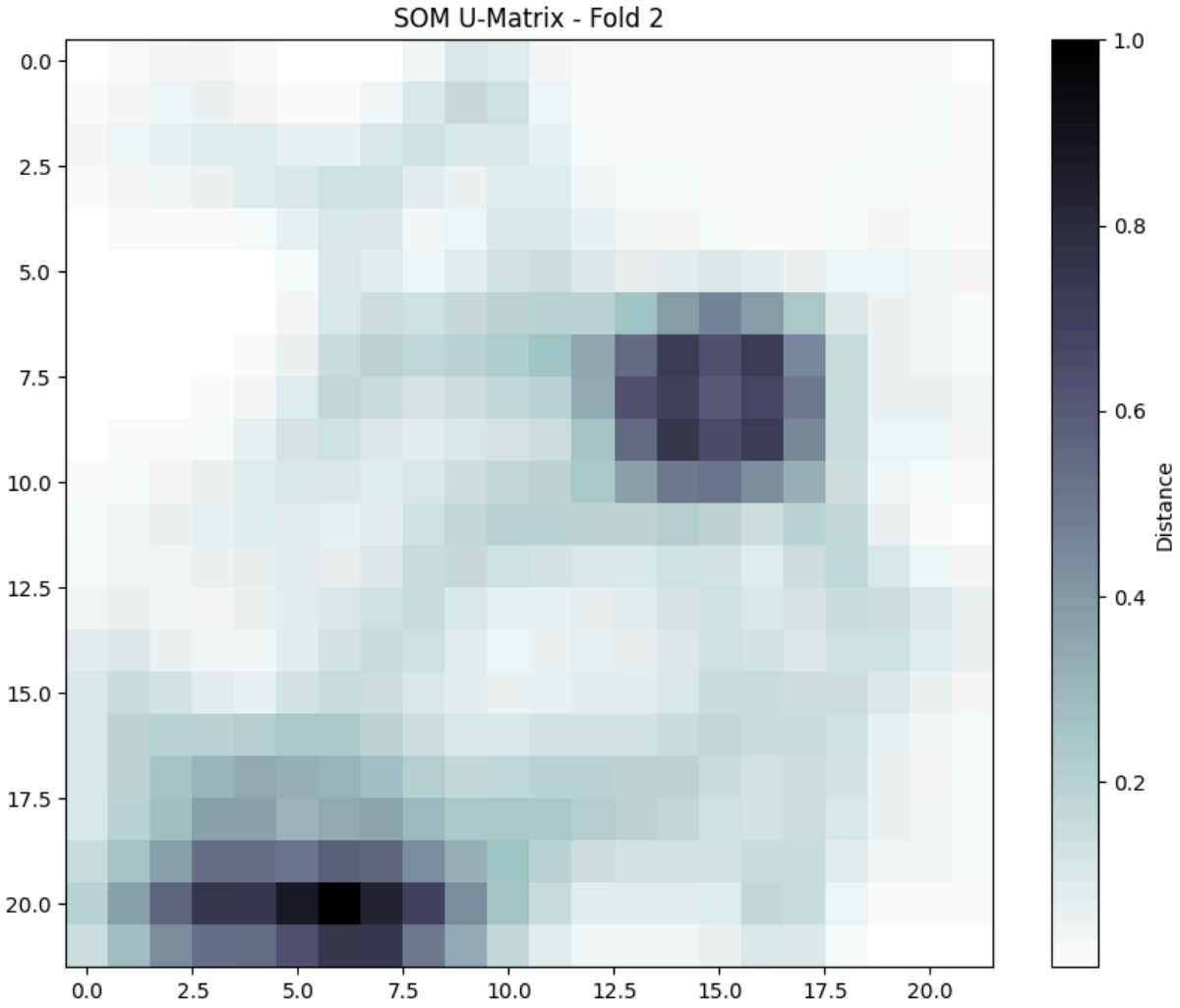
```
    plt.title("SOM ROC ")
    plt.show()

    prec, rec, _ = precision_recall_curve(oof_true, oof_score)
    PrecisionRecallDisplay(precision=prec, recall=rec).plot()
    plt.title("SOM PR")
    plt.show()

    # end of resource monitoring
    overall_end_time = time.time()
    overall_end_ram = process.memory_info().rss / 1024 / 1024  # in MB
    overall_end_cpu = psutil.cpu_percent(interval=1)

    # results
    print("\n====== SOM Validation Summary ======")
    for i, a in enumerate(accuracies, 1):
        print(f"Fold {i}: {a:.4f}")
    print(f"Mean Accuracy: {np.mean(accuracies):.4f}")
    print(f"Standard Deviation: {np.std(accuracies):.4f}")

    # resources usesd
    print("\n Overall Training Stats ")
    print(f"Total Training Time: {overall_end_time - overall_start_time:.2f} second
    print(f"Total RAM Usage Increase: {overall_end_ram - overall_start_ram:.2f} MB"
    print(f"CPU Usage (at final check): {overall_end_cpu}%")

    return accuracies
```

In [4]:

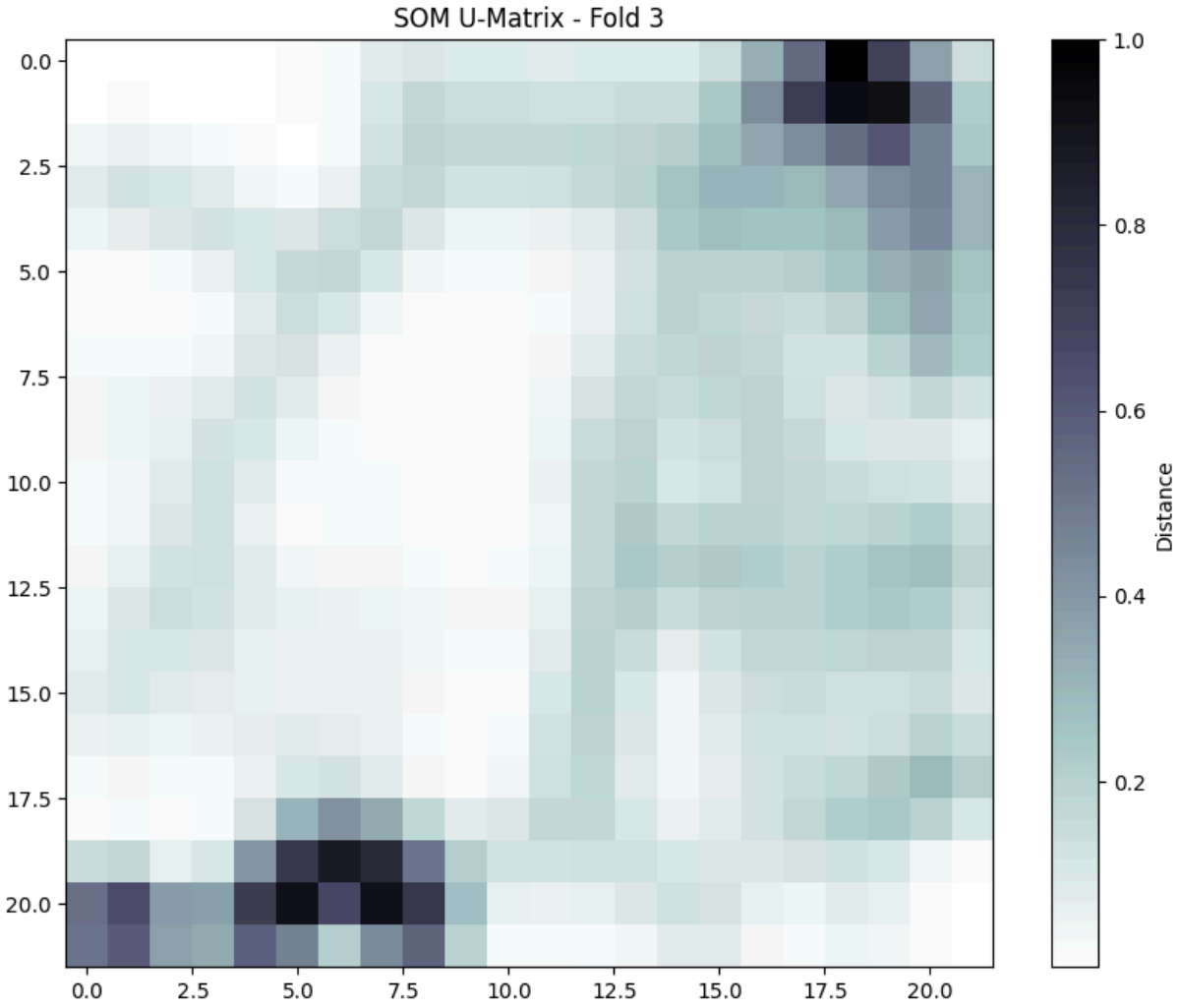
```
if __name__ == "__main__":
    # Load the dataset
    Syn_df = pd.read_csv("D:\Coding Projects\Detection-of-SYN-Flood-Attacks-Using-M

    # select first 12 feature columns (exclude label and fold info)
    feature_columns = Syn_df.columns.difference(["Label", "Fold"]).tolist()[:12]

    # run cross-validation using a Self-Organizing Map
    accs = som_cross_validate(Syn_df, feature_columns)
    print("\nFinal SOM Cross-Validation Results:")
    print(f"Fold Accuracies: {accs}")
```



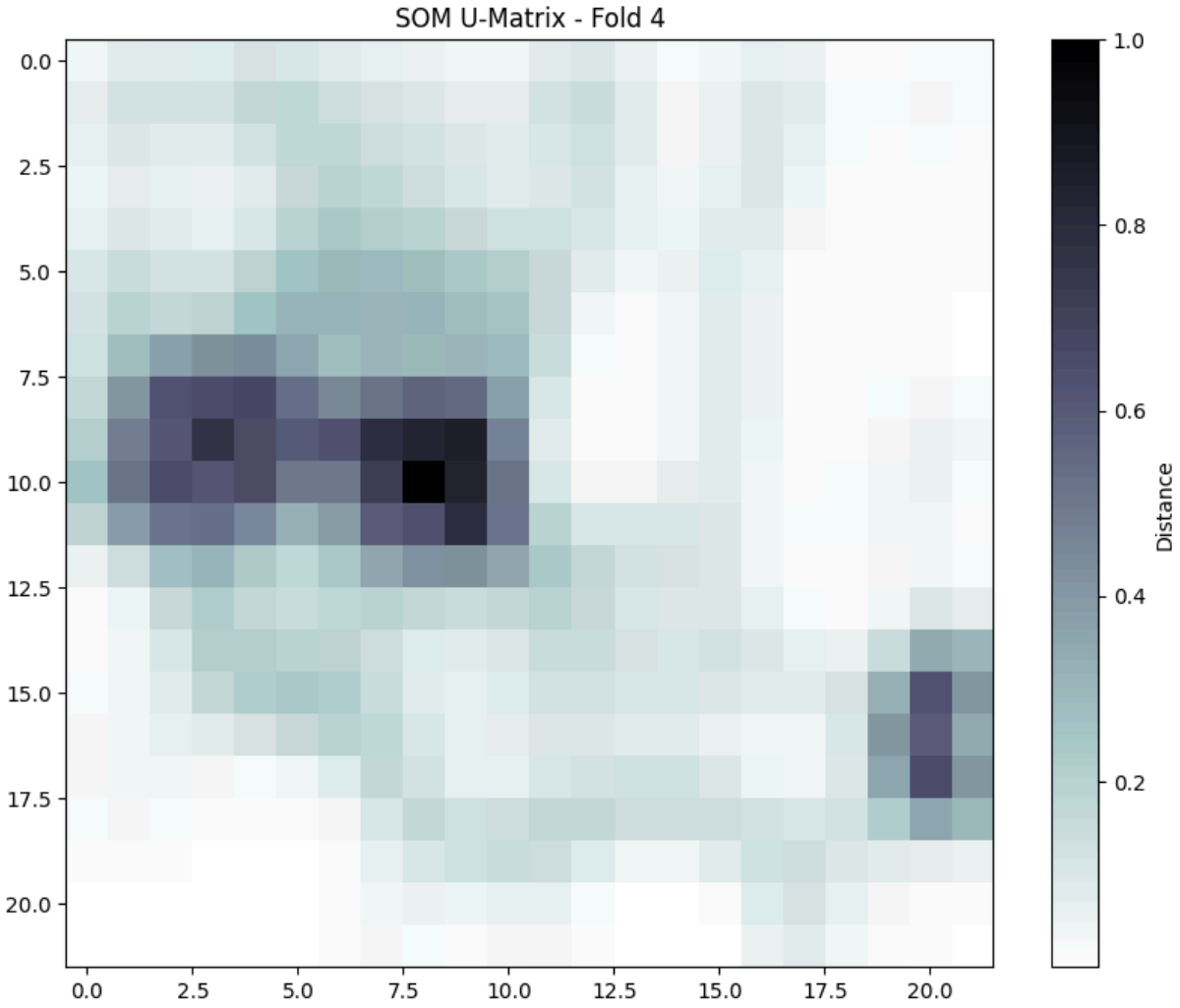

```
Fold 1: accuracy = 0.9979
```



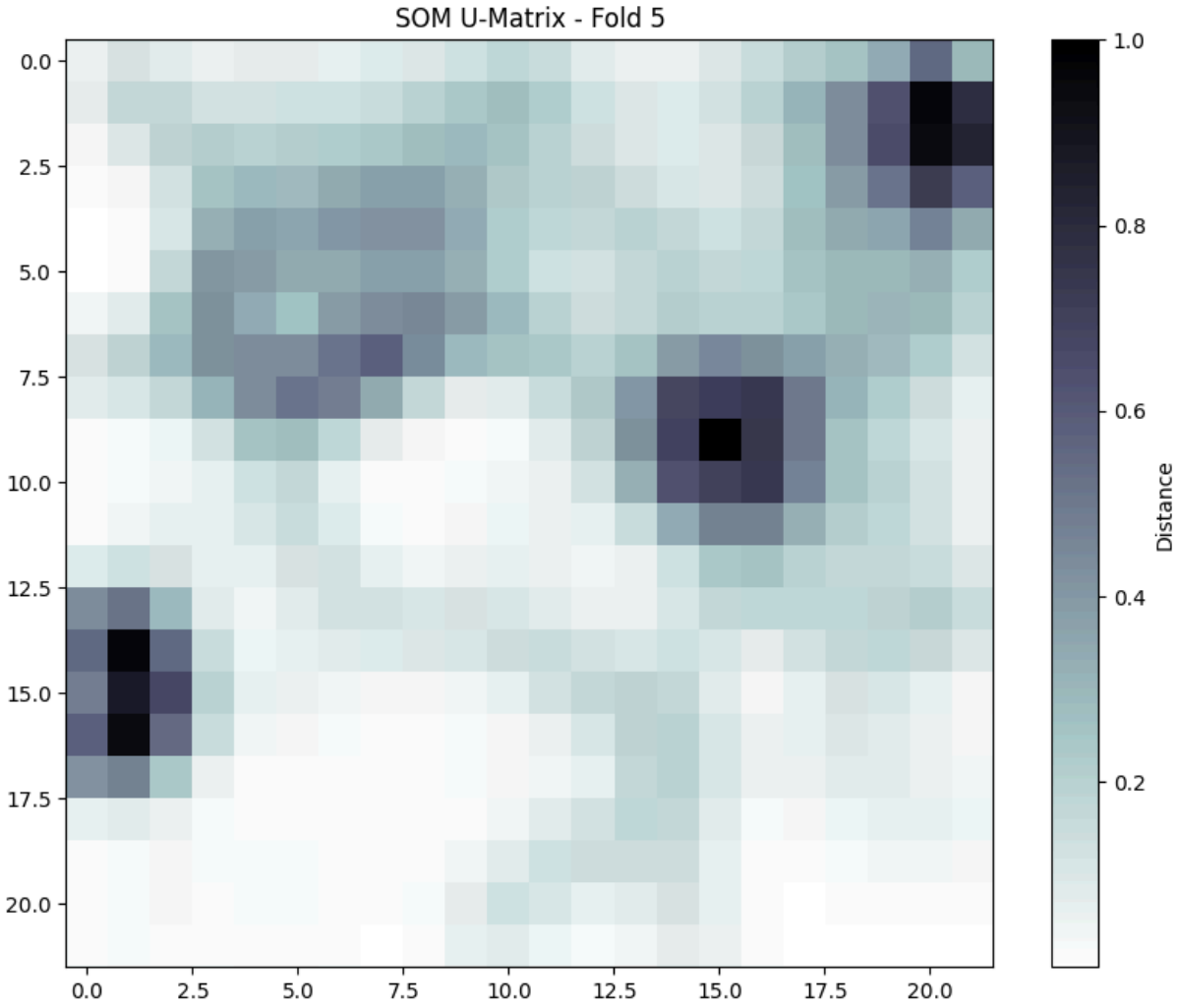
Fold 2: accuracy = 0.9984



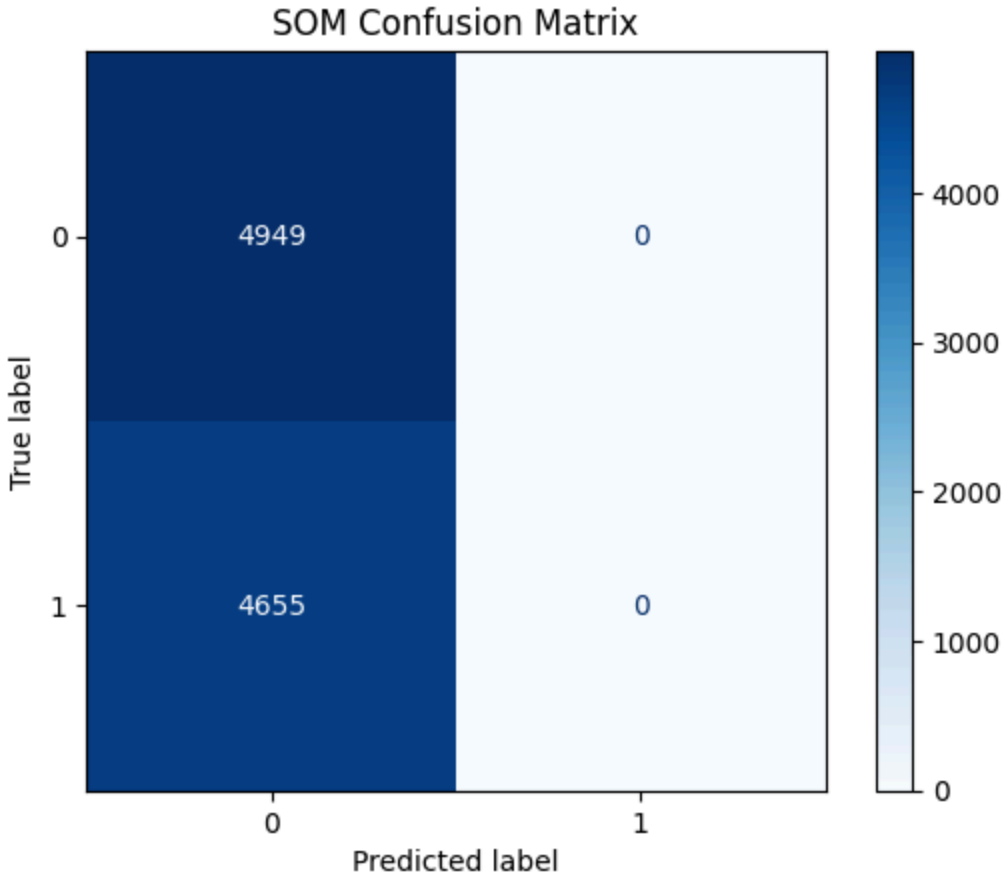

```
Fold 3: accuracy = 0.9958
```

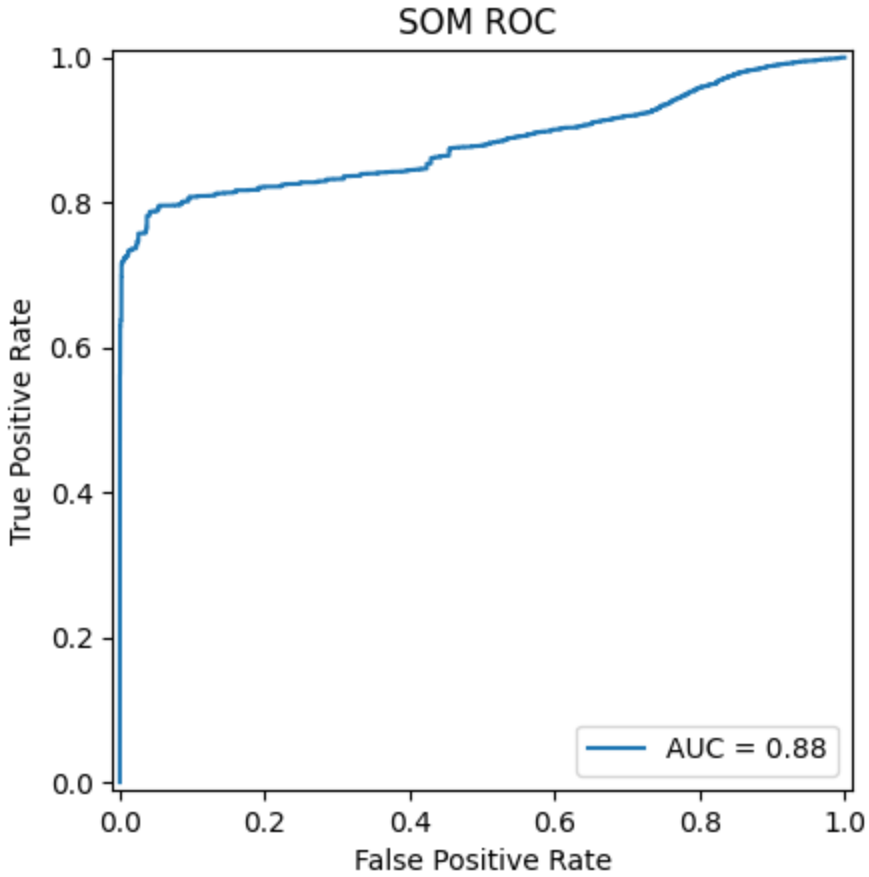


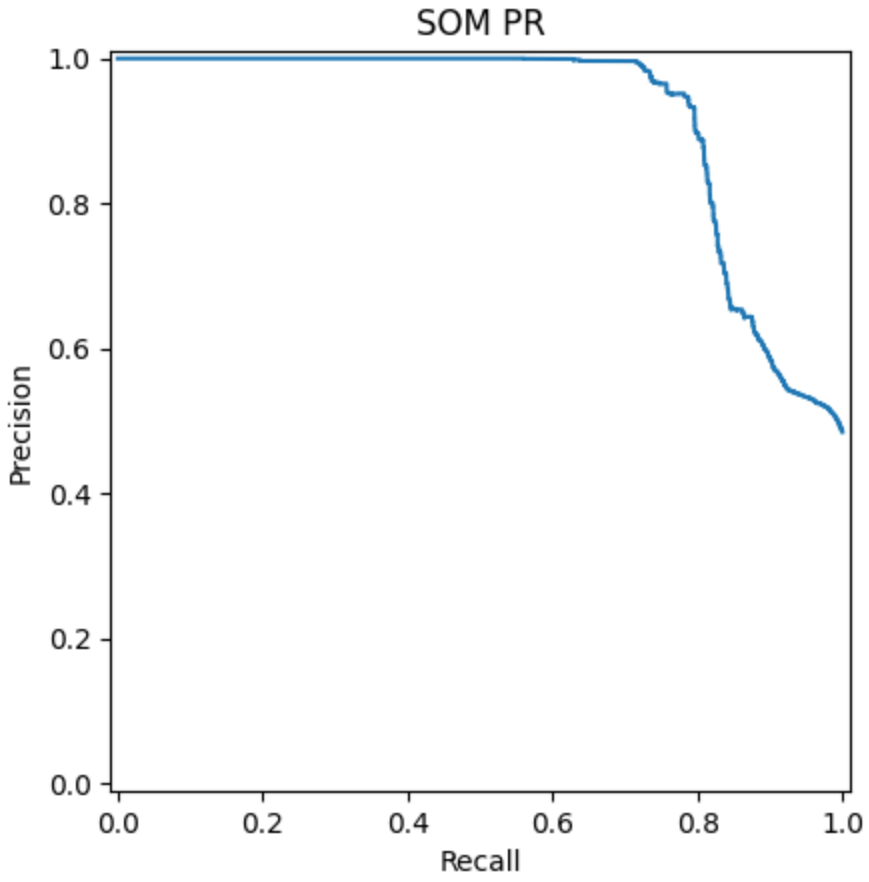
Fold 4: accuracy = 0.9974




```
Fold 5: accuracy = 0.9964
```


SOM Confusion Matrix
4949
0
4655
0
0
1
True label
0
1
Predicted label
4000
3000
2000
1000
0


SOM ROC
True Positive Rate
False Positive Rate
AUC = 0.88
0.0
0.2
0.4
0.6
0.8
1.0




```
═══ SOM Validation Summary ═══
Fold 1: 0.9979
Fold 2: 0.9984
Fold 3: 0.9958
Fold 4: 0.9974
Fold 5: 0.9964
Mean Accuracy: 0.9972
Standard Deviation: 0.0010

 Overall Training Stats
Total Training Time: 6.66 seconds
Total RAM Usage Increase: 31.69 MB
CPU Usage (at final check): 10.7%

Final SOM Cross-Validation Results:
Fold Accuracies: [0.9979177511712649, 0.9984383133784487, 0.9958355023425299, 0.9973
971889640812, 0.9963541666666667]
```

saving the model as PDF

In [5]:
```
import os
os.getcwd()
```

Out[5]:
```
'd:\\Coding Projects\\Detection-of-SYN-Flood-Attacks-Using-Machine-Learning-and-De
ep-Learning-Techniques-with-Feature-Base\\Taulant Matarova'
```

In [ ]:
```
!jupyter nbconvert --to webpdf "d:\\Coding Projects\\Detection-of-SYN-Flood-Attacks
```

```
This application is used to convert notebook files (*.ipynb)
        to various other formats.

        WARNING: THE COMMANDLINE INTERFACE MAY CHANGE IN FUTURE RELEASES.

Options
=======
The options below are convenience aliases to configurable class-options,
as listed in the "Equivalent to" description-line of the aliases.
To see all configurable class-options for some <cmd>, use:
    <cmd> --help-all

--debug
    set log level to logging.DEBUG (maximize logging output)
    Equivalent to: [--Application.log_level=10]
--show-config
    Show the application's configuration (human-readable format)
    Equivalent to: [--Application.show_config=True]
--show-config-json
    Show the application's configuration (json format)
    Equivalent to: [--Application.show_config_json=True]
--generate-config
    generate default config file
    Equivalent to: [--JupyterApp.generate_config=True]
-y
    Answer yes to any questions instead of prompting.
    Equivalent to: [--JupyterApp.answer_yes=True]
--execute
    Execute the notebook prior to export.
    Equivalent to: [--ExecutePreprocessor.enabled=True]
--allow-errors
    Continue notebook execution even if one of the cells throws an error and include
the error message in the cell output (the default behaviour is to abort conversion).
This flag is only relevant if '--execute' was specified, too.
    Equivalent to: [--ExecutePreprocessor.allow_errors=True]
--stdin
    read a single notebook file from stdin. Write the resulting notebook with defaul
t basename 'notebook.*'
    Equivalent to: [--NbConvertApp.from_stdin=True]
--stdout
    Write notebook output to stdout instead of files.
    Equivalent to: [--NbConvertApp.writer_class=StdoutWriter]
--inplace
    Run nbconvert in place, overwriting the existing notebook (only
            relevant when converting to notebook format)
    Equivalent to: [--NbConvertApp.use_output_suffix=False --NbConvertApp.export_for
mat=notebook --FilesWriter.build_directory=]
--clear-output
    Clear output of current file and save in place,
            overwriting the existing notebook.
    Equivalent to: [--NbConvertApp.use_output_suffix=False --NbConvertApp.export_for
mat=notebook --FilesWriter.build_directory= --ClearOutputPreprocessor.enabled=True]
--coalesce-streams
    Coalesce consecutive stdout and stderr outputs into one stream (within each cel
l).
    Equivalent to: [--NbConvertApp.use_output_suffix=False --NbConvertApp.export_for
```

```
mat=notebook --FilesWriter.build_directory= --CoalesceStreamsPreprocessor.enabled=Tr
ue]
--no-prompt
    Exclude input and output prompts from converted document.
    Equivalent to: [--TemplateExporter.exclude_input_prompt=True --TemplateExporter.
exclude_output_prompt=True]
--no-input
    Exclude input cells and output prompts from converted document.
            This mode is ideal for generating code-free reports.
    Equivalent to: [--TemplateExporter.exclude_output_prompt=True --TemplateExporte
r.exclude_input=True --TemplateExporter.exclude_input_prompt=True]
--allow-chromium-download
    Whether to allow downloading chromium if no suitable version is found on the sys
tem.
    Equivalent to: [--WebPDFExporter.allow_chromium_download=True]
--disable-chromium-sandbox
    Disable chromium security sandbox when converting to PDF..
    Equivalent to: [--WebPDFExporter.disable_sandbox=True]
--show-input
    Shows code input. This flag is only useful for dejavu users.
    Equivalent to: [--TemplateExporter.exclude_input=False]
--embed-images
    Embed the images as base64 dataurls in the output. This flag is only useful for
the HTML/WebPDF/Slides exports.
    Equivalent to: [--HTMLExporter.embed_images=True]
--sanitize-html
    Whether the HTML in Markdown cells and cell outputs should be sanitized..
    Equivalent to: [--HTMLExporter.sanitize_html=True]
--log-level=<Enum>
    Set the log level by value or name.
    Choices: any of [0, 10, 20, 30, 40, 50, 'DEBUG', 'INFO', 'WARN', 'ERROR', 'CRITI
CAL']
    Default: 30
    Equivalent to: [--Application.log_level]
--config=<Unicode>
    Full path of a config file.
    Default: ''
    Equivalent to: [--JupyterApp.config_file]
--to=<Unicode>
    The export format to be used, either one of the built-in formats
            ['asciidoc', 'custom', 'html', 'latex', 'markdown', 'notebook', 'pdf',
'python', 'qtpdf', 'qtpng', 'rst', 'script', 'slides', 'webpdf']
            or a dotted object name that represents the import path for an
            ``Exporter`` class
    Default: ''
    Equivalent to: [--NbConvertApp.export_format]
--template=<Unicode>
    Name of the template to use
    Default: ''
    Equivalent to: [--TemplateExporter.template_name]
--template-file=<Unicode>
    Name of the template file to use
    Default: None
    Equivalent to: [--TemplateExporter.template_file]
--theme=<Unicode>
    Template specific theme(e.g. the name of a JupyterLab CSS theme distributed
```

```
    as prebuilt extension for the lab template)
    Default: 'light'
    Equivalent to: [--HTMLExporter.theme]
--sanitize_html=<Bool>
    Whether the HTML in Markdown cells and cell outputs should be sanitized.This
    should be set to True by nbviewer or similar tools.
    Default: False
    Equivalent to: [--HTMLExporter.sanitize_html]
--writer=<DottedObjectName>
    Writer class used to write the
                                        results of the conversion
    Default: 'FilesWriter'
    Equivalent to: [--NbConvertApp.writer_class]
--post=<DottedOrNone>
    PostProcessor class used to write the
                                        results of the conversion
    Default: ''
    Equivalent to: [--NbConvertApp.postprocessor_class]
--output=<Unicode>
    Overwrite base name use for output files.
                Supports pattern replacements '{notebook_name}'.
    Default: '{notebook_name}'
    Equivalent to: [--NbConvertApp.output_base]
--output-dir=<Unicode>
    Directory to write output(s) to. Defaults
                                    to output to the directory of each notebook. To re
cover
                                    previous default behaviour (outputting to the curr
ent
                                    working directory) use . as the flag value.
    Default: ''
    Equivalent to: [--FilesWriter.build_directory]
--reveal-prefix=<Unicode>
    The URL prefix for reveal.js (version 3.x).
            This defaults to the reveal CDN, but can be any url pointing to a copy
            of reveal.js.
            For speaker notes to work, this must be a relative path to a local
            copy of reveal.js: e.g., "reveal.js".
            If a relative path is given, it must be a subdirectory of the
            current directory (from which the server is run).
            See the usage documentation
            (https://nbconvert.readthedocs.io/en/latest/usage.html#reveal-js-html-sl
ideshow)
            for more details.
    Default: ''
    Equivalent to: [--SlidesExporter.reveal_url_prefix]
--nbformat=<Enum>
    The nbformat version to write.
            Use this to downgrade notebooks.
    Choices: any of [1, 2, 3, 4]
    Default: 4
    Equivalent to: [--NotebookExporter.nbformat_version]

Examples
--------
```

```
    The simplest way to use nbconvert is

            > jupyter nbconvert mynotebook.ipynb --to html

            Options include ['asciidoc', 'custom', 'html', 'latex', 'markdown', 'not
ebook', 'pdf', 'python', 'qtpdf', 'qtpng', 'rst', 'script', 'slides', 'webpdf'].

            > jupyter nbconvert --to latex mynotebook.ipynb

            Both HTML and LaTeX support multiple output templates. LaTeX includes
            'base', 'article' and 'report'.  HTML includes 'basic', 'lab' and
            'classic'. You can specify the flavor of the format used.

            > jupyter nbconvert --to html --template lab mynotebook.ipynb

            You can also pipe the output to stdout, rather than a file

            > jupyter nbconvert mynotebook.ipynb --stdout

            PDF is generated via latex

            > jupyter nbconvert mynotebook.ipynb --to pdf

            You can get (and serve) a Reveal.js-powered slideshow

            > jupyter nbconvert myslides.ipynb --to slides --post serve

            Multiple notebooks can be given at the command line in a couple of
            different ways:

            > jupyter nbconvert notebook*.ipynb
            > jupyter nbconvert notebook1.ipynb notebook2.ipynb

            or you can specify the notebooks list in a config file, containing::

                c.NbConvertApp.notebooks = ["my_notebook.ipynb"]

            > jupyter nbconvert --config mycfg.py

To see all available configurables, use `--help-all`.

[NbConvertApp] WARNING | pattern 'd:\\\\Coding Projects\\\\Detection-of-SYN-Flood-At
tacks-Using-Machine-Learning-and-Deep-Learning-Techniques-with-Feature-Base\\\\Taula
nt Matarova\\\\SOM model.ipynb' matched no files
```